So you’ve chosen Jenkins to ease and automate your work, but you’re having some issues with its performance. This is often the case if you don’t keep track of some essential things, such as Jenkins best practices, updates, plugins, and some useful tricks to make your work faster and more efficient.

What is Jenkins and What Are Its Benefits for Business?

Jenkins is a Java-written open source automation tool that is built with plugins for continuous integration purposes. Aside from that, it can serve as a continuous delivery hub for any project.

Jenkins:

1. makes it easier for software developers to integrate changes to the project and for users to obtain a fresh build;
2. allows you to continuously deliver your software by integrating with numerous testing and deployment technologies;
3. enables organizations to speed up the software development process through automation;
4. integrates the development of life-cycle processes, such as build, document, test, package, stage, deploys, static analysis, and so on.

Step 1: Jenkins Performance

While Jenkins is a powerful tool, it is not infallible. Sometimes implementations run into performance issues that slow down builds, or even completely halt projects. It can force the team to scramble into troubleshooting mode. However, there are measures that you can take for Jenkins performance tuning. By taking these precautionary measures, you can at the very least smooth processes out, if not entirely eradicate performance issues at each level of the project. If you initiate configuration changes early on and apply Jenkins CI/CD best practices, your Jenkins workloads will have better performance for a longer period.

Step 2: Updates and Backups

Jenkins regularly releases new versions. For Jenkins performance tuning, it is advised that you always make use of the latest Long-Term Support (LTS) release and shun the updates in-between. The same holds for plugins. These do not have Long-Term Support releases, but several updates can occur. Generally, if it is working properly, then it’s better to not update it. Some updates could work well on one device and cause issues on another, so avoid unknown updates and versions to be on the safe side.

To ensure stability, it is paramount that you create backups of your Jenkins server. It is possible to back up all data, but that will cost you a lot in terms of time and resources. On the other hand, you can only back up your configuration files via scripts.

Step 3: Minimize Number of Builds on Master Nodes

A master node refers to where the application will be running. It is the ‘brain’ of your Jenkins, and it is not replaceable, unlike a slave. Therefore, for Jenkins performance tuning, you must free the master nodes from unnecessary tasks. Doing so will leave the CPU and memory to be utilized for scheduling and triggering builds on slaves only.

Step 4: Minimize Build-up History

When configuring a job, you define the number of builds and the period for which they will be left in the file system before they get deleted. This feature, Discard Old Builds, becomes quite critical when you trigger numerous builds of that job in a short timeframe.

If the history limit is too high, it means too many builds will be kept in the file system. In such circumstances, Jenkins will need too many old builds, for example, to display them in the history widget. It will be executed very slowly, particularly if you are trying to open those job pages. With this in mind, for Jenkins performance tuning, you should limit the number of builds that you keep to reasonable levels.

Step 5: Consider Your Plugins Again

Jenkins has a long catalog of plugins. It is easy to get confused with its variety. When dealing with Jenkins, it is crucial to decide whether or not you need a particular plugin. Plugins have the potential to negatively affect the functionality within Jenkins in ways you would not have expected. They are also known to slow down your jobs during building. It is necessary to make the correct plugin selection for a particular job.

Keeping track of installed plugins are among best Jenkins practices. Uninstalling any plugins that are no longer necessary helps keep Jenkins bloat-free. Some plugins come with global configuration settings. It is crucial to highlight that some default settings may not be efficient for Jenkins performance tuning. Therefore, you should make sure they are in alignment with the requirements for Jenkins’s optimal performance.

Step 6: Tool Integration

Jenkins CI/CD best practices include integration Jenkins with other tools, such as Gitlab, Github, Gerrit, or any bugtracker which can boost performance. REST API makes it possible to integrate external tools in Jenkins. You can easily combine Jenkins with Gitlab or Github, whichever you prefer. Every time a software developer creates a pull request to your source code, Jenkins can be triggered to check the pull request for any faults. It is critical to ensure that you only approve requests that comply with the standards.

**Gerrit**

When reviewing the code within Gerrit, you will keep getting feedback from Jenkins. You must check if everything adheres to your specifications, and if so, you can automatically merge the code in several branches.

**Bug Trackers**

Another approach is to integrate Jenkins with bug trackers, such as Jira or Redmine. For example, one can automatically create a changelog based on all the reported issues within this particular build.

It is crucial to highlight that other integrations are possible. For instance, you can easily deploy your application on each build. An automated integration will ensure that you can work on other issues rather than performing those tasks manually.

Step 7: Define a Heap Size

Defining the right heap size is relevant to Jenkins as it is to any Java application. Most of the modern Java apps get started with a maximum heap size configuration. There is a very crucial JVM feature you should be aware of when defining the heap size. This feature is known as UseCompressedOops, and it functions on widely used 64-bit platforms. This feature shrinks the objects’ pointers from 64 bit to 32 bit, saving a lot of memory in the process. When defining the ideal heap size, it is recommended that you stay below 32GB.

Step 8: Set Up Correct Slaves

As part of your Jenkins best practices, it is recommended that you only incorporate easy-to-manage slaves. You should be able to replace or add new ones without difficulty. It is critical that it only takes a couple of clicks to get a new slave up and running if a slave crashes. It is enabled by creating virtual images that you can use to install or manage your complete configuration via [Chef or Puppet](https://opsworks.co/devops-configuration-management-tools/).

Another strategy that you can use for Jenkins performance tuning is to make slaves generic. It enables you to run a myriad of jobs on a single slave, making it more useful. Putting slaves on the cloud enables you to activate them automatically with a container, such as Docker, for instance. This way, you save resources, and it also allows you to scale up automatically when more slaves are required.

Step 9: Tune the Garbage Collector

The garbage collector is a powerful automatic memory management process. Its prime purpose is to identify the unused objects that will be in the heap and free up the memory that they hold. It is important to highlight that some of the garbage collector actions cause the Java application to freeze. It often happens when your application holds a heap of more than 4 Gigabytes. In such cases, garbage collector tuning is necessary to shorten the pause time.

Step 10: Apply Multiple Jenkins Masters

If you have several teams or projects, you might utilize multiple Jenkins masters. For instance, you can create a master for PHP projects and another one for Android projects. It will ensure that changes do not have an effect on all projects and that you have only the required plugins on a particular master for each project.

Step 11: Split Jenkins Jobs

As a part of your Jenkins best practices, you can also split your job, as it eliminates the risk of failures occurring towards the end of a long build. Splitting jobs into smaller jobs enables you to easily restart if one of them fails. There is a plugin called Workflow that is very powerful for this purpose.

Step 12: Generate Jobs Easily

If you have numerous projects, then you should consider generating jobs automatically. Jenkins utilizes the powerful REST API and CLI tools for this purpose. You can also make use of plugins like Workflow that automate processes. There is the Cloudbees templates plugin which allows you to generate jobs from a predefined job skeleton easily.

Step 13: Plan the Jenkins Disk Usage Growth

It is a common occurrence to run out of disk space as your Jenkins installation gobbles up more resources. To improve Jenkins performance, one of the most pivotal aspects is considering the disk to brace for the inevitable growth of disk usage. This is due to the fact that Jenkins usage can grow swiftly, especially when you begin to host numerous jobs.

Step 14: Mind your Storage

Storage is inexpensive; therefore, you should ensure that you have adequate space in the short term. It is also crucial to highlight that it is difficult to estimate the required storage upfront. Therefore, the best approach is to ensure that you can grow later, which usually requires upfront planning. It is not necessary to spend money on expensive SCSI disks because, with Jenkins, there is no need for rapid disk access. Most of the disk space is utilized for the storage of bits that are rarely accessed.

Step 15: Clear Old Jenkins Data

Jenkins CI/CD best practices include applying the old data management feature. Jenkins maintains jobs and builds on the file system. When you perform an action, such as upgrading your core, installing or upgrading a plugin, then the data format might be altered. Under such circumstances, Jenkins keeps the old data format in the file system and loads the previous data format to the memory. This is extremely beneficial if you have to roll back your upgrade. However, there are some cases when it is unnecessary to have too much data loaded to the memory. High memory consumption is associated with slow user interface responsiveness or even OutOfMemory errors. To minimize such occurrences, it is advised that you open the old data management page (http://JenkinsUrl/administrativeMonitor/OldData/manage), confirm that the data is not required, and delete it.

Step 16: Be Part of Jenkins Community

Jenkins has a huge community behind it: thousands of documents, tips, how-tos, etc. It is beneficial to you to become actively involved in the Jenkins community by either writing, contributing plugins, or solving issues that will help improve the Jenkins experience. Such a move ensures that you are always up to date with Jenkins best practices and all of the latest developments.